2017 IEEE 14th International Conference on Mobile Ad Hoc and Sensor Systems

Study of Internet-of-Things Messaging Protocols used for Exchanging Data with External Sources

Ajay Chaudhary, Sateesh K. Peddoju, and Kavitha Kadarla
High Performance Computing Lab
Department of Computer Science & Engineering
Indian Institute of Technology Roorkee
Roorkee, India
ajaychaudhary@acm.org, sateesh@ieee.org, satkavi.23@gmail.com

***Abstract*—Internet-of-Things (IoT) is emerging as one of the popular technologies influencing every aspect of human life. The IoT devices equipped with sensors are changing every domain of the world to become smarter. In particular, the majorly benifited service sectors are agriculture, industries, healthcare, control & automation, retail & logistics, and power & energy. The data generated in these areas is massive requiring bigger storage and stronger compute. On the other hand, the IoT devices are limited in processing and storage capabilities and can not store and process the sensed data locally. Hence, there is a dire need to integrate these devices with the external data sources for effective utilisation and assessment of the collected data. Several existing well-known message exchange protocols like Message Queuing Telemetry Transport (MQTT), Advanced Message Queuing Protocol (AMQP), and Constrained Application Protocol (CoAP) are applicable in IoT communications. However, a thorough study is required to understand their impact and suitability in IoT scenario for the exchange of information between external data sources and IoT devices. In this paper, we designed and implemented an application layer framework to test and understand the behavior of these protocols and conducted the experiments on a realistic test bed using wired, wifi and 2/3/4G networks. The results revealed that MQTT and AMQP perform well on wired and wireless connections whereas CoAP performs consistently well and is less network dependent. On the lossy networks, CoAP generates low traffic as compared to MQTT and AMQP. The low memory footprint of MQTT and CoAP has made them a better choice over AMQP.**

***Keywords*-IoT; Internet of Things; MQTT; AMQP; COAP; MQTT-SN; IoT Protocols; Application Layer Protocols; IoT applications;**

I. Introduction

In the past few years, Internet of Things (IoT) has gained increasing attention because of their capability to provide novel and attractive solutions in the areas such as agriculture [1]–[3], industrial automation, environmental monitoring, transportation, healthcare [4], and disaster management. The IoT is deployed almost everywhere we can think off. They are deployed in a range from tens to thousands with or without a physical wired link and layout. They are deployed in an unstructured or structured fashion. These are inexpensive as compared to their traditional counterparts. They can sense pressure, motion, temperature, humidity, chemical, biosensors, luminosity, gyroscope, gasses (such as Co_2 and O_2), acoustic and position. They also help in the air, water, and soil quality monitoring. These devices are also widely deployed for monitoring of habitat, an active volcano, and biodiversity. They are also helpful in monitoring hidden dangers of underground mines including oxygen saturation or unstable mine structure. It is used for ocean monitoring to determine related critical aspects and helps fishermen to find the better fishing area. These are to list a few. Since these are inexpensive as compared to traditional sensors, they can be deployed in large scale in any application leading to the massive production of data. Due to their limited processing and storage capacity, they require transmitting sensed data towards the server or gateway. The traditional communication and application protocols do not fit well within IoT environments due to the limited processing capabilities. For efficient transmission of data towards storage or processing clusters such as Cloud computing and High-Performance Computing (HPC) facilities, the IoT devices must be equipped with the constrained IoT application layer protocols. The overall success of IoT depends on well-defined application layer protocols.

II. BACKGROUND: IOT APPLICATION LAYER PROTOCOLS

The advancement in mobile devices and communication systems provided a rapid momentum for IoT based solutions. With the rapid increase in overall utilization of IoT enabled systems, there is a rapid growth in the amount of data captured. Moreover, data generated needs to be managed according to its requirements, to create more valuable services. For this purpose, integration of WSN or IoT with cloud computing is becoming very essential. Figure 1 depicts the generic model of communication and data exchange between IoT and external storage sources like Cloud and HPC cluster. The figure indicates several of application layer protocols used in the communications. Keeping the characteristics of IoT devices such as the large area deployment, minimal power consumption, and weak computational capabilities,

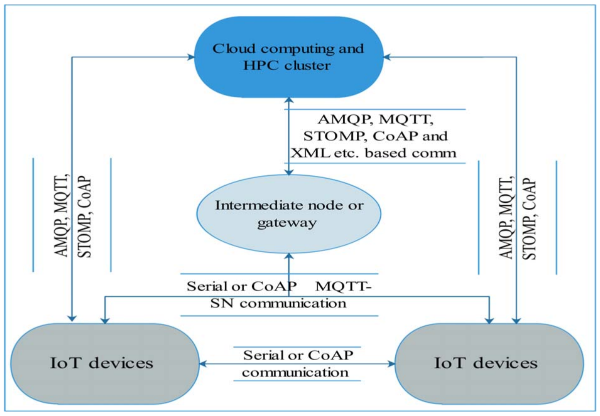
Cloud computing and HPC cluster
AMQP, MQTT, STOMP, CoAP and XML etc. based comm
AMQP, MQTT, STOMP, CoAP
AMQP, MQTT, STOMP, CoAP
Intermediate node or gateway
Serial or CoAP MQTT-SN communication
IoT devices
Serial or CoAP communication
IoT devices

Figure 1. Model of IoT devices and HPC or cloud communication

there is a great need for efficient application layer protocols. They must be well-tuned for seamless data transfer from IoT devices to the external sources. These protocols are expected to have the following properties.

- It is necessary to fit the protocol object code into the little memory of IoT devices.
- The protocols must not consume too much power to process data and send it to the external source.
- The protocols must be suitable for interoperable environments. To that extent, they need to follow the standards.
- The protocols must be open source to get widespread acceptance and usability.

To provide information exchange services to a wide group of IoT devices efficiently, these protocols must deliver services in a publisher/subscriber model which is simple to operate. The client who wants data must register service request through a broker and is popularly known as a *subscriber*. In the same way, some clients constantly produce data; they must register as a *publisher* with the broker. The publisher/subscriber models work on the paradigm that publisher constantly produces the data/service and subscriber constantly consumes the data/service. The overall architecture is straightforward and easy to implement. Some of the well-known data exchange protocols are: Message Queuing Telemetry Transport(MQTT) [5], Message Queuing Telemetry Transport for sensor Network(MQTT-SN) [6], Constrained Application Protocol(CoAP) [7], Extensible Messaging and Presence Protocol (XMPP) [8], Data Distribution Service(DDS) [9], Advanced Message Queuing Protocol(AMQP) [10] and Simple Text Oriented Message Protocol(STOMP) [11]. Out of these protocols, MQTT, AMQP, and COAP are widely used. These protocols are discussed in detail in the following subsections.

Message Queuing Telemetry Transport (MQTT) [5]: It is a one of most commonly used publisher/subscriber based protocol. The overall working of MQTT is very simple where client subscribes for a topic with broker and publisher sends data related to that specific topic to the broker, and subscriber client constantly consumes data associated with the same topic. The topic is in a hierarchical format. MQTT works on TCP protocol, and hence it provides reliable services. Moreover, to provide additional safety, it provides three QoS features.

1) *QoS 0*: In this, the message may be delivered once or not at all.
2) *QoS 1*: In this, the message is delivered to the receiver at least once.
3) *QoS 2*: In this, the message is delivered exactly once without duplicate or loss.

MQTT also support alive feature which helps to provide a reliable connection throughout the period of transmission.

Advanced Message Queueing Protocol (AMQP) [10]: It is one of the recently proposed protocols arising from the financial industry. It is highly useful for the exchange of bulk messages as compared to the RESTful Web services. By making use of this protocol, massive amounts of data can be sent securely to the data store without significantly influencing the performance of the whole system. It is an open standard for Message Oriented Middleware (MOM) communication. AMQP middleware supports different applications to send and receive messages irrespective of platform and programming languages used. Though it can support different transport protocols, it assumes an underlying reliable transport protocol such as TCP. In AMQP, the messages are self-contained, and data contained in a message is opaque and immutable. AMQP provides various ways to transfer messages such as *Publish-and-subscribe*, *point-to-point* and *store-and-forward*.

An AMQP messaging system consists of these main components [12]:

1) *Publisher*: An application that generates messages and sends to the AMQP broker.
2) *Consumer/Subscriber*: An application that receives the messages from one or more publishers through a broker.
3) *Broker/Server*: A server or a daemon program that contains one or more Virtual Hosts, Exchanges, Message Queues, and Bindings.
4) *Virtual Host*: A daemon application program, having services such as Hosts, Exchanges, Message Queues, and Bindings.

Each of these components run on autonomous hosts or machines.

Constrained Application Protocol (CoAP) [7]: It is intended to provide better services to the IoT devices which are mostly resource constrained to run the application. It is based on REpresentational State Transfer (REST) architecture. It uses Universal Resource Identifier (URI) to communicate between server and client. The communication between client and server is through response request model. The CoAP protocol runs one UDP stack. It is a specialized web transfer protocol for use with constrained nodes and constrained networks which are low powered, and lossy. The CoAP is a solution for integration of low power IoT network with web services. CoAP has also been adapted by the operating systems like Contiki and TinyOS. The CoAP protocol is specially designed for machine-to-machine (M2M) applications. The CoAP provides a request/response interaction model between application endpoints, similar to IPv6 over Low-Power Wireless Personal Area Networks (6LoWPANs). It supports the built-in discovery of services and resources, provides a feature like asynchronous message exchanging, and includes key concepts of the Web such as URIs. CoAP proposes to use Datagram Transport Layer Security (DTLS) as the security protocol for automatic key management, data encryption, and integrity protection as well as for authentication. The CoAP protocol consists of the following main components:

- *Endpoint*: It is the source or destination of a CoAP message. It is identified by IP address and a UDP port number.
- *Sender*: The source endpoint that generates messages.
- *Recipient*: The destination endpoint that consumes messages.
- *Client*: A client is similar to a sender or recipient, but it can place a request instead of the message.
- *Server*: The destination endpoint of a request; the originating endpoint of a response.
- *Origin Server*: The server on which a given resource resides or is to be created.
- *Intermediary*: A CoAP endpoint that acts both as a server and as a client towards an origin server. A common form of an intermediary is a proxy.
- *Proxy*: It helps in packet forwarding, provides namespace support, and also does protocol translation.

III. Literature survey

There are several works in which the AMQP, CoAP, and MQTT protocols evaluated. Subramoni et al. [12] used Apache Qpid as AMQP broker. They tested the performance of AMQP by simulating financial transactions of the stock exchange. They used Infiniband and gigabit Ethernet network to test their Qpid based AMQP middleware. They evaluated the performance of AMQP based on the matrix parameters like publishers/consumers and type of data exchange, message exchange rate, and message latency. They found that the centralized architecture of AMQP is acting as a bottleneck for achieving excellent stability. Bandyopadhyay et al. [13] compared both MQTT and COAP models and derived a relation between MQTT overhead and packet size. They used WANEM to emulate the LAN and Unix based system to run the protocols. They compared protocols based on their specific characteristics and also tried to find bandwidth resource tradeoff. To test their platform, they sent different sized packets over the same communication channel to the same destination. They simulated link characteristics to generate different network conditions. Based on their model, they found that COaP is efficient regarding the energy consumption as well as bandwidth utilization. They also observed that gateways suffer due to processing speed.

Thangavel et al. [14] designed a middleware and simulated the network characteristics. They tried to find the end-to-end delay and bandwidth of two most common IoT protocols, MQTT and CoAP. They compared the protocol performance of COAP and MQTT using a laptop, a BeagleBoard-XM and a netbook as the server, publisher, and subscriber respectively. They connected BeagleBoard-xM to netbook through a wired interface on which the WANEM emulator is running. They estimated latency and overhead at different loss rates. They found that at low packet loss rate, MQTT had a lower delay compared to CoAP. However, on high packet loss rates, MQTT has a significant delay as compare to CoAP. Luzuriaga et al. [15] compared AMQP and MQTT for information exchange over the unstable and mobile networks. They Evaluated AMQP and MQTT based on network parameters like latency, jitter, and saturation boundary values, and message queuing.

Yokotani and Sasaki [16] evaluated the MQTT and HTTP protocols, and estimated the bandwidth usage and delay depending on power consumption by the device. They compared the MQTT and HTTP protocols based on network overhead with zero application load. They found that the MQTT results in more number of data transfers compared to HTTP. The number of transmitted bytes depends on the number of devices connected to a server. MQTT provides a light load to the server as compared to the HTTP. Then they suggested the enhancements to MQTT to improve its overall performance. Chen and Kunz [17] compared Data Distribution Service (DDS) using OpenDDS with MQTT using HiveMQ and Mosquitto, CoAP using CCoAP (Californium CoAP). They found that DDS outperformed MQTT in term of telemetry latency in a network scenario in term of high network packet loss, high latency, and narrow bandwidth.However, DDS consumes high bandwidth. MQTT comparatively is more reliable than DDS.

Viswanathan [18] analysed the power profiling of MQTT protocol. The author implemented MQTT on Raspberry Pi-III board to test power consumption using packets of various sizes. Tested the broker based on more than one publishers and found that a single broker can serve the increased number of publishers. , However, due to high processing overhead, its consumes more power in it. It is observed that in a stable network, the data transferred is low while in the lossy network, the amount of data transferred is high.It is also found that the authentication process does not add any additional overhead over the power consumption, but it

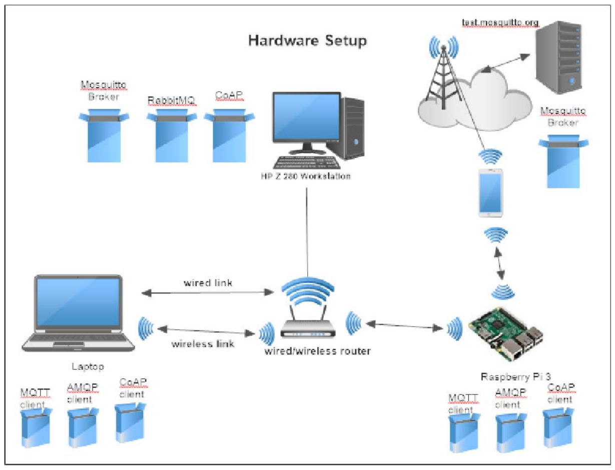

Figure 2. Hardware setup and networking scheme of the framework

lowers the transfer rate of the data.

Several authors tried to evaluate AMQP, CoAP, and MQTT, but not altogether in the same experimental setup. Most of them attempted to simulate the protocols using network simulators such as WANEX. These scenarios never reflect the real working environments. In our work, we tested the system in a realistic setup where the IoT devices are connected through wireless, wired and 4/3/2 G networks. in which, the IoT devices are deployed with weak or meager signal strength and other networking and environmental factors like interferences. Our experimental setup captures real environmental issues in more detail, and same is reflected in results.

IV. Experimental Setup

The publish/subscribe, or client/server scenario helps us to setup experiment on different autonomous device/machines. The various components of the test bed are deployed on various machines/devices having a significant variation of processing speeds and power consumptions. The broker/server is setup on a high-end machine with high processing speed. The overall architecture is supported multiple protocols and also provides a gateway to access the desired service. On a client machine/device, we setup common service access platforms for each of AMQP, MQTT, and CoAP. The AMQP and MQTT are setup with TCP protocol at network layer while CoAP with UDP protocol. The CoAP supports IPv6, and its services are accessible through URI. To get a service on any topic, first, we generate a URI for that service. For generating URI on the required topic, IPv6/Ipv4 address and port number are passed to the common service access platform and, in turn, it returns an accessible URI. The client passes URI with a request to the server. The server posts the message to the client. In our setup, the both client and server are setup with IPv4 addresses. We used C based libcoap library [19] to deploy the CoAP client and server. The MQTT is tested using Mosquitto [20] broker deployed on local machine connected through wired/wireless LAN and also using *test.mosquitto.org* [21] broker. The MQTT is based on the publisher/subscriber service framework. In our setup, there are multiple publishers/subscribers. We varied the load based on packet size as well as based on publishers/subscribers. The AMQP is also a broker based messaging service. The publishers/subscribers access the AMQP protocol to publish/subscribe the messages. It stores and forwards the messages, unlike other protocols. The messages are delivered to the subscriber while at the same time, a copy of the message is stored on the broker. The RabbitMQ [22] broker is deployed for AMQP services. We deployed AMQP and MQTT publisher/subscriber using Python and CoAP client/server using C and shell script program to generates URI to access the required services. For testing proposes, we generated and delivered the messages in quick successions and also varied the load of the packet for statical analysis purposes. Here, the messages are generated in multiple of 10, i.e., 10, 100, 1000, 10000, etc. We used the Wireshark [23] to capture and analyze the data. Figure 2 shows the general architectural framework of components used to run the applications.

The HPC Z820 server is setup to run RabbitMQ broker/server [22], Mosquitto broker [20], and Libcoap server [19]. Wireshark [23] is used to capture and analyze the data is installed on the same machine. Raspberry Pi-III is used to running MQTT client, libcoap client, and AMQP client programs. The application programs are written in Python, based on AMQPs Python pika [24] and MQTT python paho [25] library. Pi-III is connected to the workstation through the wireless medium and also connected to the *test.mosquitto.org* [21] to get a trace of 4/3/2G networks with modest signal strength. The HP Pavilion Notebook laptop is connected through wire / wireless medium to the workstation. We ran and tested the client programs for MQTT, CoAP and AMQP on HP Pavilion Notebook laptop to get Wireshark traces and these traces are used and compared with the trace generated by the client programs running on the Pi-III.

V. Results and Discussion

In our experimental setup, we try to create the various possible setups to understand the overall working of MQTT, AMQP and COAP protocol in a more broader sense. We tested the protocols on a reliable network like wired and unreliable network like wireless, 4/3/2G networks with the low signal strength to get the realistic situation. Using following different metrics, we estimated the behavior of MQTT, AMQP, and CoAP protocols.

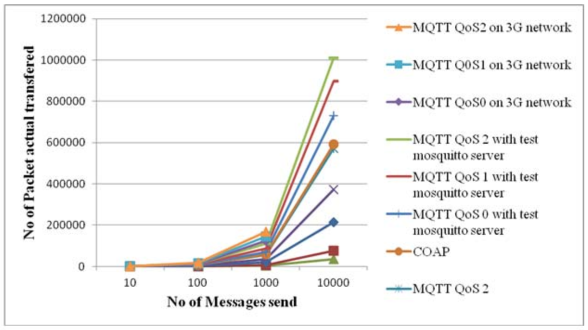

Figure 3. No. of Packet transferred

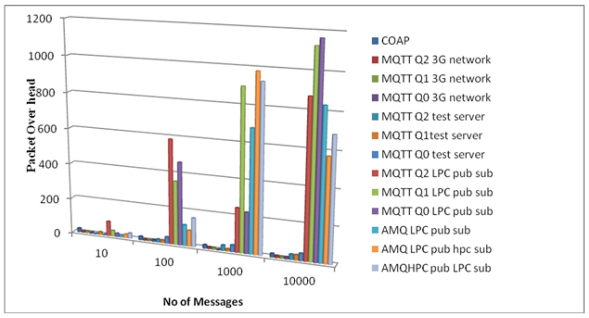

Figure 4. Packet Overhead

A. Packet Overhead

Packet overhead is the number of packets generated in the network by the MQTT, COAP and AMQP protocols to deliver the messages. In this experiment, we tried to assess the impact of packet overhead on those three protocols.The figures 3 and 4 refer to the total number of packets generated by the client and server or broker during the test phase and packet overhead due to the extra packets generated to deliver the messages. As we can see, due to QoS requirement, the packet overhead due to MQTT is high. We can also see that the lossy 3G mobile network has highest packet overhead ratio due to a large number of packet drops. In the case of low traffic, there is no significant distinguishable factor to determine packet overhead as most of the time packets are delivered without or fewer packet losses.

B. Message Throughput

Figure 5 depicts the overall throughput of the messages. Here, the message throughput is the number of messages sent per unit time. As we can notice, for low volumes, almost all the protocols generated a right amount of messages and shown equivalent message throughput. However, for high volumes, the MQTT delivered messages exactly once to meet its QoS requirements giving high throughput. It generated lots of packets to be sent to the network. It might have happened due to the packet losses or retransmission of the packets.

C. Bandwidth Utilization

Figure 6 shows the overall bandwidth utilization by each protocol.The figure infers the amount of bandwidth utilization required for each protocol. At high message rate, the bandwidth utilization of CoAP and MQTT QoS 1 and QoS 2 level is highest. They flooded the network with lots of packets to deliver the packets to the broker/server. When the services need to be availed from the remote locations, for example in applications like agriculture, the deployment of MQTT with QoS 1 and QoS 2 level is not advisable using the low capacity networks like mobile networks (for example, 2G and 3G networks).

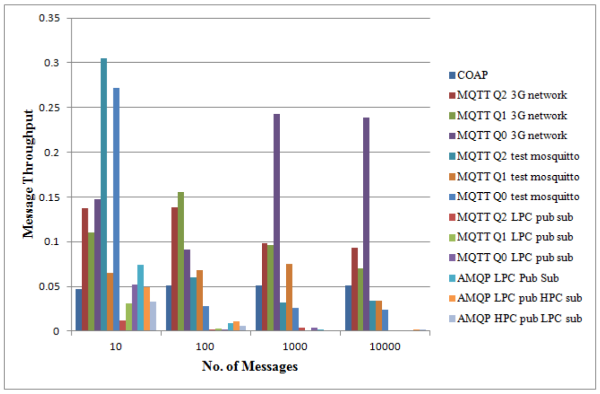

Figure 5. Message throughput

VI. Conclusions

The results revealed that the MQTT and CoAP perform well on wired and wireless connection whereas CoAP performs consistently throughout and is less network dependent. Also at high packet rates, MQTT with QoS 1 and QoS 2

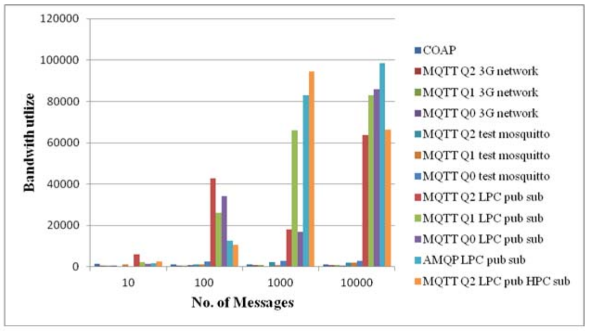

Figure 6. Bandwidth Utilize

levels excluding QoS 0 flooded the network with the packets due to a large number of retransmissions. However, it is observed that due to its small memory footprint and enough open source community support, it has become a favourable protocol for IoT applications. CoAP is excellent protocol for constrained applications but its overall model is based on client/server approach which limits its use for large scale deployments. AMQP is not well-suited protocol for IoT applications deployed over ESP8266 due to its large memory footprint and hardly available public libraries for the device like Arduino, ESP8266. It still needs open source community support to make it possible for low memory devices.

References

[1] P. S. Barath, M. Dutta, A. Chaudhary, and M. S. Jangid, "A novel adaptive framework for efficient and effective management of water supply system using arduino," in *Proceedings of the International Conference on Information and Communication Technology for Competitive Strategies 2014 (ACM ICTCS '14)*, ser. ICTCS '14. Udaipur, Rajasthan, India: ACM, 2014, pp. 13:1–13:4.

[2] A. Chaudhary, "A cluster based wireless sensor network deployment for precision agriculture in dried and arid states of india," in *Proceedings of the International Conference on Information and Communication Technology for Competitive Strategies, 2014 (ACM ICTCS '14)*, ser. ICTCS '14. Udaipur, Rajasthan, India: ACM, 2014, pp. 60:1–60:3.

[3] L. Atzori, A. Iera, and G. Morabito, "The internet of things: A survey," *Computer Networks*, vol. 54, no. 15, pp. 2787 – 2805, 2010.

[4] A. Chaudhary, S. K. Peddoju, and S. K. Peddoju, "Cloud based wireless infrastructure for health monitoring," in *Cloud Computing Systems and Applications in Healthcare*. IGI Global, 2017, pp. 19–46.

[5] A. Banks and R. Gupta, "Mqtt version 3.1. 1," *OASIS standard*, vol. 29, 2014.

[6] U. Hunkeler, H. L. Truong, and A. Stanford-Clark, "Mqtt-s a publish/subscribe protocol for wireless sensor networks," in *Proceedings of the 3rd International Conference on Communication Systems Software and Middleware and Workshops, 2008 (IIEEE COMSWARE'08)*, Bangalore, India, Jan 2008, pp. 791–798.

[7] Z. Shelby, K. Hartke, and C. Bormann, "The constrained application protocol (coap)," 2014.

[8] (2017) XMPP protocol specification. [Online]. Available: http://xmpp.org/

[9] G. Pardo-Castellote, "Omg data-distribution service: architectural overview," in *Proceedings of the 23rd International Conference on Distributed Computing Systems Workshops, 2003*, Providence, Rhode Island, USA, USA, May 2003, pp. 200–206.

[10] (2017) AMQP protocol specification. [Online]. Available: https://www.amqp.org/

[11] (2017) STOMP protocol specification. [Online]. Available: https://stomp.github.io/

[12] H. Subramoni, G. Marsh, S. Narravula, P. Lai, and D. K. Panda, "Design and evaluation of benchmarks for financial applications using advanced message queuing protocol (amqp) over infiniband," in *Proceedings of the Workshop on High Performance Computational Finance, 2008*, Austin, TX, USA, Nov 2008, pp. 1–8.

[13] S. Bandyopadhyay and A. Bhattacharyya, "Lightweight internet protocols for web enablement of sensors using constrained gateway devices," in *Proceedings of the International Conference on Computing, Networking and Communications (IEEE ICNC'13)*, San Diego, CA, USA, Jan 2013, pp. 334–340.

[14] D. Thangavel, X. Ma, A. Valera, H. X. Tan, and C. K. Y. Tan, "Performance evaluation of mqtt and coap via a common middleware," in *roceedings of the 9th International Conference on Intelligent Sensors, Sensor Networks and Information Processing (IEEE ISSNIP'14)*, Singapore, Singapore, April 2014, pp. 1–6.

[15] J. E. Luzuriaga, M. Perez, P. Boronat, J. C. Cano, C. Calafate, and P. Manzoni, "A comparative evaluation of amqp and mqtt protocols over unstable and mobile networks," in *2015 12th Annual IEEE Consumer Communications and Networking Conference (CCNC)*, Jan 2015, pp. 931–936.

[16] T. Yokotani and Y. Sasaki, "Comparison with http and mqtt on required network resources for iot," in *2016 International Conference on Control, Electronics, Renewable Energy and Communications (IEEE ICCEREC'16)*, Bandung, Indonesia, Sept 2016, pp. 1–6.

[17] Y. Chen and T. Kunz, "Performance evaluation of iot protocols under a constrained wireless access network," in *roceedings of the International Conference on Selected Topics in Mobile Wireless Networking (IEEE MoWNeT'16)*, Cairo, Egypt, April 2016, pp. 1–7.

[18] A. Viswanathan, "Analysis of power consumption of the mqtt protocol," June 2017. [Online]. Available: http://d-scholarship.pitt.edu/32399/

[19] O. Bergmann, "libcoap: C-implementation of coap," *http://libcoap. sourceforge.net*, 2017.

[20] (2017) Mosquitto MQTT server/broker. [Online]. Available: test.mosquitto.org/

[21] R. Light. (2017) Mosquitto-an open source mqtt v3. 1 broker. [Online]. Available: http://mosquitto.org/

[22] A. Videla and J. J. Williams, *RabbitMQ in action: distributed messaging for everyone*. Manning, 2012.

[23] (2017) Wireshark. [Online]. Available: http://www. wireshark. org/

[24] (2017) AMQP for python (pika). [Online]. Available: https://github.com/pika/pika/

[25] (2017) Python software foundation. paho-mqtt 1.1. [Online]. Available: https://pypi.python.org/pypi/paho-mqtt/